POLITECNICO DI MILANO
Corso di Laurea in Ingegneria Informatica
Dipartimento di Elettronica e Informazione

RoboWII 2.0: un gioco interattivo con un robot autonomo

AI & R Lab
Laboratorio di Intelligenza Artificiale
e Robotica del Politecnico di Milano

Relatore: Prof. Andrea Bonarini

Tesi di Laurea di:
Antonio Micali, matricola 700016

Anno Accademico 2008-2009

Ringraziamenti

Ringrazio i miei genitori che con i loro sacrifici mi hanno permesso di raggiungere questo primo obiettivo riponendo in me la loro fiducia e per l'esempio di vita che continuano a darmi. Ringrazio pure mia sorella per la sua pazienza dimostrata tutte le volte che per i miei esperimenti ho invaso i suoi spazi.
Ringrazio il professor Andrea Bonarini per avermi seguito durante lo svolgimento del lavoro con consigli che mi hanno fatto intraprendere, ogni volta, le scelte più appropriate.
Desidero poi ringraziare tutte le persone che lavorano in AirLab. In particolare Simone Ceriani e Davide Rizzi per la continua disponibilità e prontezza nei chiarimenti e suggerimenti.
Un ringraziamento ad Antonio Bianchi e Ben Chen che hanno sviluppato la prima versione del progetto che mi è stata un utile punto di partenza.
Infine un abbraccio a tutti gli amici dell'università con i quali ho condiviso questi tre anni di intenso studio (ma anche di piacevoli svaghi).

Indice

Capitolo 1

Introduzione

L'interazione tra robot autonomi e uomo é in questo periodo ambito di numerose ricerche. Con l'avanzare della scienza e della tecnologia i rapporti tra robot e uomini cambieranno notevolmente.
Questa tesi si colloca nell'area dell'interazione tra uomo e robot ed ha l'obiettivo di realizzarne uno in grado di giocare con le persone.

1.1 Obiettivi

L'uscita della console Nintendo Wii ha cambiato il modo di giocare con le console. L'utilizzo del Wii Remote (di seguito abbreviato in Wiimote) fornisce molti spunti per giochi sempre più interattivi e multimediali.
Lo scopo della tesi è quello di creare un nuovo gioco: un robot autonomo che riesca a interagire con l'uomo e con l'ambiente esterno. Il gioco si interfaccia con il controller della consolle Wii ed è studiato per permettere di giocare anche in un ambiente ostile come può essere il salotto di un appartamento.

1.2 Breve descrizione del lavoro

Il gioco è stato creato scegliendo un robot commerciale a basso costo (Spykee della Meccano) e adattandolo per poter interagire con l'uomo e con l'ambiente.
Lo scopo del gioco creato è quello di colpire un robot che deve, invece, evitare di farsi centrare. In un ambiente quale può essere quello del salotto si trovano numerosi ostacoli che complicano i movimenti del robot. Quindi, nel muoversi, il robot deve stare attento a tavoli, sedie, persone che passano

o qualsiasi altro oggetto si trovi sulla sua traiettoria, modificandola di conseguenza.
RoboWii 2.0 è stato sviluppato in ambiente Linux tramite il linguaggio ad oggetti C++.
Per la comunicazione con il Wiimote è stata utilizzata la libreria open source Wiiuse che permette un immediato e alquanto semplice uso del dispositivo della Nintendo.
Sono stati, inoltre, utilizzati i seguenti software precedentemente utilizzati dal Politecnico di Milano all'interno del progetto MRT (Milano Robocap Team):
- DCDT, per lo scambio di messaggi all'interno dei vari componenti di RoboWii.
- Mr. Brian, per la programmazione dei comportamenti del robot tramite logica fuzzy.

1.3 Risultati

Il risultato finale è stato quello di ottenere un robot in grado, tramite sensori opportunamente montati e integrati su di esso, di interagire con l'uomo: scappando dal giocatore tentando di non essere colpito, evitando gli ostacoli e si accorgendosi di essere stato colpito.

1.4 Struttura della tesi

Di seguito viene elencata e descritta brevemente la struttura della tesi:
- nel primo capitolo viene fatto un inquadramento generale dell'area di applicazione del progetto
- nel secondo capitolo si descrivono le tecnologie che si sono adottate
- nel terzo capitolo si parla dello svolgimento del progetto
- nel quarto capitolo si descrive il gioco e i comportamenti del robot
- nel quinto e ultimo capitolo si analizzano i risultati e si racconta dei possibili miglioramenti
- l'appendice A riporta l'analisi della comunicazione con Spykee
- nell'appendice B si riporta il manuale del software prodotto

Capitolo 2

Software e tecnologie utilizzate

2.1 Spykee

Spykee è un robot prodotto dalla Meccano che ha le sembianze di un mini robot umanoide, due cingoli al posto delle gambe, due braccia decorative ai lati ed una videocamera sulla sommità.
Il robot, dotato di scheda Wi-Fi / 802.11 b/g con il quale può comunicare con un PC, è venduto corredato dell'apposito software di controllo che permette di gestirne ogni funzionalità. Se connesso ad un router Wi-Fi è possibile anche controllare il robot da un qualsiasi computer connesso ad internet. Spykee è dotato di una videocamera con una risoluzione di 320x240 pixel e riesce a trasmettere fino a 15 fotogrammi al secondo. Le immagini sono compresse in formato jpeg e poi inviate al pc. Spykee è anche dotato di un riproduttore mp3, un microfono ambientale e un po' di luci con finalità estetiche.

Figura 2.1: Spykee, il robot spia.

Purtroppo al momento dell'inizio del progetto non erano disponibili librerie che gestissero le funzioni del robot. Per questo si è dovuto catturare i pacchetti contenenti i dati inviati al robot e interpretarli.

Inoltre questo robot non dispone di nessuna forma di odometria, e non è quindi possibile stimare i movimenti delle ruote cingolate.

2.2 Wii Remote

Il Wii Remote, chiamato normalmente Wiimote, è il principale controller della console Wii della Nintendo. La principale caratteristica del Wiimote è la sua capacità di rilevamento del movimento che permette agli utenti di interagire con i giochi attraverso il movimento e il puntamento, tramite l'uso di accellerometri e sensori ottici.
Il Wii Remote è stato annunciato al Tokyo Game Show il 16 settembre 2005. Da lì ha sempre attratto molta attenzione per le sue uniche caratteristiche e la sua differenza rispetto ai precedenti controller.

Figura 2.2: Wii Remote.

Visto il suo basso costo e per le sue caratteristiche, il Wiimote è molto usato anche per controllare dispositivi non legati alla consolle della Nintendo.

Al suo interno, il Wiimote contiene:
- un accelerometro a tre assi ADXL330
- un sensore ottico PixArt con un filtro per far passare solo le frequenze dell'infrarosso
- 10 tasti digitali standard, più un tasto power e un tasto sync
- uno speaker
- 4 led
- un piccolo motore elettrico, collegato ad una massa eccentrica, che gli consente di vibrare
- una memoria EEPROM di 16 KB

Il Wiimote è alimentato tramite 2 batterie AA ed è pensato per un utilizzo wireless comunicando tramite il protocollo bluetooth. Inoltre, è presente una porta di espansione alla quale è possibile collegare componenti aggiuntivi, ad esempio un controller analogico, un volante o una finta chitarra.
Per utilizzare il Wiimote attraverso il PC sono nate diverse librerie per la sua gestione. Quella usata in questo progetto e la Wiiuse [4].

2.2.1 La telecamera IR

Sulla parte frontale del wiimote è montato un sensore Pixart che svolge la funzione di telecamera. Più precisamente si tratta di una telecamera monocromatica con definizione 128x96 con applicato un filtro che rende visibile al Wiimote solo luci infrarosse (quindi con una lunghezza d'onda attorno a 940 nm). Integrato nel Wiimote e collegato alla telecamera è presente un processore che si occupa di tracciare il movimento di quattro punti. Questo significa che il Wiimote non considera l'intera immagine acquisita dalla telecamera, ma solo i punti tracciati dal processore. Per tracciare l'andamento di questi punti è applicata una 8x subpixel analysis, che rende la dimensione spaziale di 1024x768. Il campo visivo della telecamera è di 45 gradi sull'asse X e di 35 gradi sull'asse Y e la frequenza di scansione è di 100Hz.
La Nintendo Wii utilizza come fonte di illuminazione una sensor bar, un dispositivo da posizionare sopra o sotto il televisore composta da due gruppi di cinque led infrarossi posti ad una distanza predefinita.

2.2.2 L'accelerometro

Il Wiimote monta un accelerometro a tre assi che sono allineati al contenitore, a meno di imprecisioni di montaggio, come mostrato in Figura 2.4. Precisamente l'ADXL330 prodotto da Analog Device. Questo accelerometro è in grado di rilevare accelerazioni fino a +/- 3g con un'incertezza del 10%.

Figura 2.3: Sensor bar montata su un televisore.

L'accelerazione è calcolata misurando lo spostamento di piccole masse presenti all'interno dell'integrato. La forza misurata su ogni asse è digitalizzata con una precisione di 8 bit.

2.2.3 Il bluetooth

La connessione tra Wiimote e altri dispositivi (tra cui il PC) avviene tramite lo standard Bluetooth HID (Human Interface Device). Il chip che provvede a gestire queste comunicazioni è il Broadcom BCM2042. Il Wiimote non necessita di nessuna autenticazione, ma bisogna solamente provvedere a metterlo in discoverable mode. Per far ciò è possibile premere il pulsante sync sul retro del dispositivo o premere contemporaneamente i tasti 1 e 2.
Una volta connesso, il Wiimote invia un report sul suo stato (tasti premuti, valori dell'accellerazione, punti infrarossi visti,...) con una frequenza massima di 100Hz. Di default un report viene inviato solo allo scatenarsi di qualche evento sul dispositivo. E' comunque possibile far sì che venga inviato con una frequenza di 100Hz.

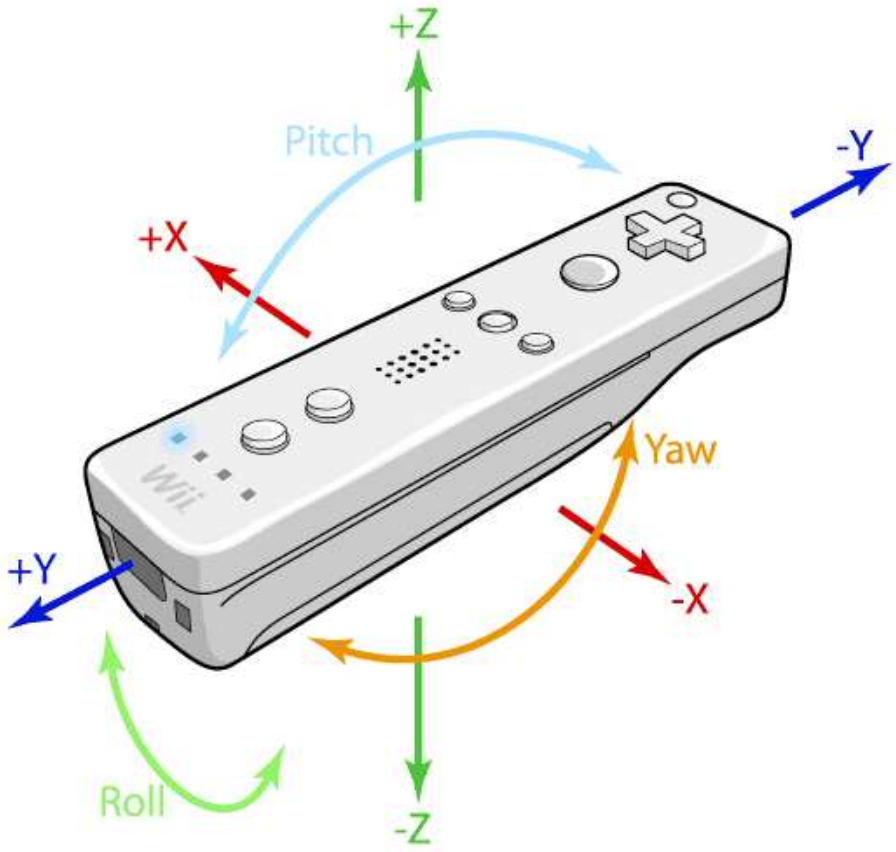

Figura 2.4: Wii Remote e gli assi lungo i quali vengono misurate le accelerazioni.

2.3 Sonar

Nati per uso sottomarino, i sonar vengono impiegati per la misurazione precisa di distanze. Grazie al grande uso in campo automobilistico, il costo di questa tecnologia è relativamente basso.
Per l'uso dei sonar è stata necessaria una scheda realizzata precedentemente in AirLab. Questa scheda monta un PIC30F4012 contenente il programma per la gestione dei sonar, il calcolo delle distanze e l'invio dei dati al computer e un MAX232 per l'interfacciamento del pic con il PC.

Figura 2.5: L'LD-MaxSonar-EZ2.

Il software risiedente nel PIC è fatto in modo che inizia la ricezione se riceve sulla seriale il carattere R e la ferma se riceve il carattere S. La scheda monta i sonar LV-MaxSonar-EZ2. Questi hanno un angolo di apertura di 30 gradi, una frequenza massima di lettura di 20Hz e sono in grado di misurare distanze che variano tra 0 e 6,45m.

2.3.1 Funzionamento del sonar

Il funzionamento dei sonar è molto intuitivo. Come si vede in figura 2.6, essi inviano un'onda sonora con una frequenza non udibile (onda rossa); quest'onda, quando incontra degli oggetti, viene riflessa e torna indietro al sonar che l'ha generata. Sapendo la velocità di propagazione delle onde sonore nell'aria (circa 343 m/s) e misurando il tempo trascorso tra l'invio e la ricezione dell'onda è possibile ricavare la distanza dell'oggetto più vicino al sonar

$$dist = 343m/s * \frac{t}{2} \tag{2.1}$$

dove dist è la distanza tra il sonar e l'oggetto più vicino e t è il tempo che l'onda sonora impiega a tornare indietro al sonar.

2.4 Vishay TSLA6400

I Vishay TSLA6400 sono dei LED che emettono la radiazione luminosa nella frequenda dell'infrarosso. I LED sfruttano le proprietà ottiche di alcuni

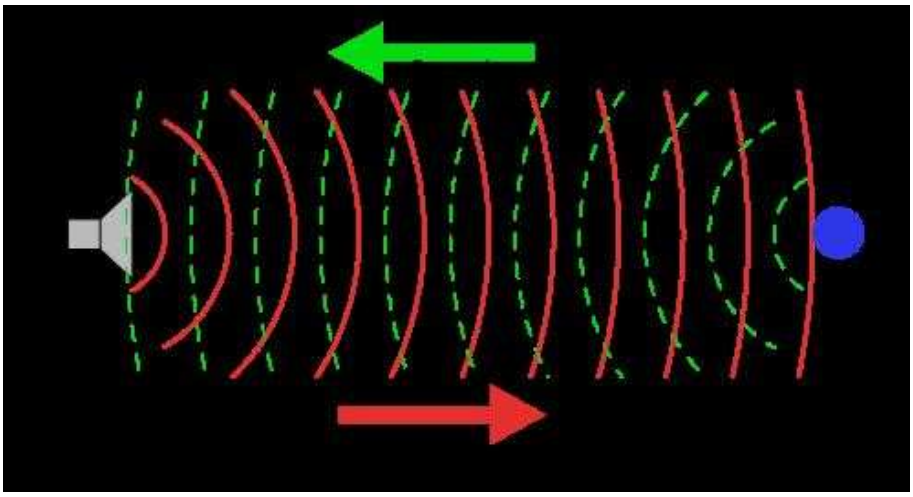

Figura 2.6: Principio di funzionamento del sonar.

materiali semiconduttori per produrre fotoni a partire dalla ricombinazione di coppie elettrone-lacuna.
I TSLA6400 sono caratterizzati dalle seguenti caratteristiche:
- lunghezza d'onda 940nm
- angolo di irradiamento 20 gradi, lo spettro è mostrato in figura 2.7 - tensione tipica ai capi 1,3V - corrente di passaggio 100mA - intensità di irradiamento 40mW/sr (con una corrente di 100mA)

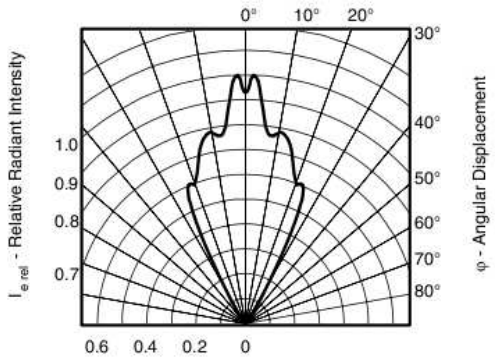

Figura 2.7: Vishay TSLA6400: Intensità radiale in funzione dell'angolo.

2.5 Moduli XBee

Per risolvere problemi di comunicazione senza fili, sono stati realizzati dalla Digi International i moduli XBee. Si tratta di moduli conformi allo standard IEEE 802.15.4 che non richiedono alcuna configurazione per funzionare e permettono uno scambio di dati affidabile tra dispositivi diversi.
Tra le caratteristiche spiccano il bitrate (250 KBps), la distanza massima di trasmissione (circa 40m) per modulo e il costo tutto sommato piuttosto accessibile: circa 22 euro.

Figura 2.8: Modulo XBee.

Come detto in precedenza i moduli XBee si servono del protocollo IEEE 802.15.4 e precisamente implementano le specifiche ZigBee. Caratteristiche di questo protocollo sono l'economicità e i bassi consumi: gli Xbee vengono mantenuti, infatti, in stato di "sleep" e quindi basso consumo energetico e vengono automaticamente attivati qualora si verifichi la necessità di una trasmissione o ricezione di dati.

2.6 Altri utilizzi di Wiimote e robot

Di seguito verranno brevemente presentati altri progetti, già realizzati, di interazione tra Wiimote e robot. La maggior parte dei robot che utilizzano il wiimote per essere controllati lo fanno semplicemente attraverso i tasti direzionali.
Un'altra classe di robot utilizza invece gli accelerometri montati sul wiimote. Con gli accelerometri è possibile ricostruire la posizione del wiimote nello spazio e quindi ricavare gli angoli di pitch, roll e yaw. Ad esempio si potrebbe controllare un robot attraverso gli angoli di pitch e di roll, il primo per farlo muovere avanti e indietro e il secondo per farlo ruotare a destra e a sinistra. Il vantaggio di

Figura 2.9: Casmobot.

questo tipo di comando rispetto al primo è che, in questo modo, possiamo avere le due rotazioni contemporanee in modo da poter muovere il robot avanti o indietro e nel frattempo farlo ruotare.
La figura 2.9 mostra un esempio di quest'uso fatto da un ricercatore danese, Kjeld Jensen. Egli ha creato un robot tagliaerba che ha la particolarità di poter essere comandato da un wiimote. In questo modo, stando seduti all'ombra e lontano dal fastidioso rumore del motore del tagliaerba è possibile comandarlo tramite il movimento, di qualche grado, della mano.

2.6.1 WiiBot

WiiBot è un altro esempio di interazione tra robot e wiimote. Questo è un braccio robotico comandato tramite gli accelerometri del wiimote. A differenza dell'utilizzo descritto in precedenza, in questo esempio il Wiimote è usato per creare una traiettoria da far eseguire al robot. Tramite gli accelerometri è possibile individuare i punti per cui è passato il wiimote e creare la traiettoria in tre dimensioni da far eseguire al robot.
Montando come end-effector una racchetta da tennis o una spada si riesce a far giocare a tennis o a scherma il robot.

2.6.2 Wiimba

Wiimba (Figura 2.10), sviluppato da Ron Tajima, è forse l'interazione tra robot e wiimote più simile a Robowii. Per questo progetto sono stati utilizzati Roomba, noto come robot aspirapolvere, e un wiimote. Su Roomba sono stati montati un ricevitore bluetooth e una barra con agli estremi dei led ad infrarossi. A questo punto il wiimote riesce a riconoscere la posizione dei led infrarossi e manda i comandi ai motori del robot facendolo muovere in modo da avere Roomba sempre puntato dal wiimote.
L'effetto che si ottiene è quello di avere un aspirapolvere senza manico.

Figura 2.10: Wiimba.

2.7 Logica fuzzy

In logica tradizionale una certa affermazione (o predicato) può essere vera o falsa.
La logica fuzzy [3] stravolge questa idea e crea gradi di verità per i predicati. Quindi un predicato non deve essere per forza o vero o falso ma può anche assumere valori in mezzo a questi due (come un po vero, tanto vero,...). Questa logica trova ampio impiego in campo robotico perché permette di programmare i robot seguendo un approcio comportamentale. In questo modo il comportamento definitivo del robot è ottenuto dalla collaborazione di vari componenti atomici e indipendenti. I comportamenti del robot sono stati implementati utilizzando questa logica. Si è utilizzato il programma Mr. Brian [1], realizzato dal Politecnico nel corso del progetto MRT. Un sistema fuzzy come Mr. Brian parte da dei dati in ingresso, li elabora secondo opportune regole e poi ritorna dei dati che sono il risultato delle regole precedenti.
L'elaborazione dei comportamenti da parte di Mr. Brian è organizzata in sei passi:
- fuzzificazione degli ingressi
- definizione dei predicati
- scelta dei comportamenti da attivare
- valutazione delle regole
- inferenza
- defuzzificazione
- valutazione di un nuovo livello di comportamenti

2.7.1 Fuzzificazione

S'intende per fuzzificazione il procedimento attraverso il quale le variabili di ingresso (come ad esempio pressioni, temperature, portate, pH, ...) vengono convertite in misure fuzzy della loro appartenenza a determinate classi come ad esempio Nulla, Bassa, Media, Alta, Molto Alta. Tale conversione da grandezze deterministiche a fuzzy viene effettuata attraverso le funzioni di appartenenza predefinite per quelle classi. In Mr. Brian è necessario indicare i tipi di variabili da fuzzificare e i nomi delle variabili in ingresso (ad esempio SonarLeft OBJECTDISTANCE indica che la variabile in ingresso SonarLeft è di tipo OBJECTDISTANCE).
Un oggetto OBJECTDISTANCE viene dichiarato in questo modo:
(OBJECTDISTANCE

(TOL (NEAR 400 500))
(TRA (MEDIUM 450 530 1000 1100))
(TOR (FAR 1150 1250))
).
Ad esempio, nel caso la variabile SonarLeft assuma il valore di 800 il valore assunto dalla funzione di NEAR sarà 0, dalla funzione di MEDIUM sarà 1 e dalla funzione di FAR sarà 0.
In figura 2.11 sono elencate le varie funzioni utilizzabili in Mr. Brian.

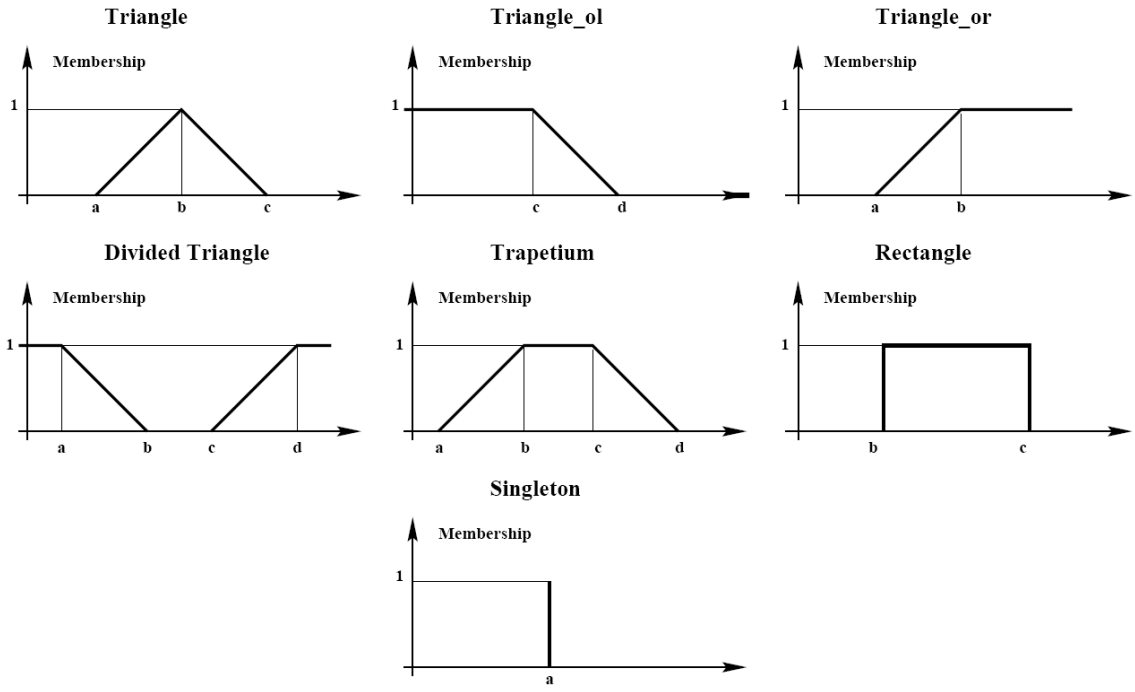

Figura 2.11: Tipi di funzini fuzzy utilizzabili in Mr. Brian.

2.7.2 Definizione dei predicati

Definite le variabili fuzzy e la loro tipologia, è necessario definire i predicati associati alle variabili in input. Un esempio di predicato è il seguente:
LeftBusy = (D SonarLeft NEAR);
LeftBusy assumerà un valore di verità compreso tra 0 e 1 che è il valore assunto dalla funzione NEAR dichiarata precedentemente. Nel caso in cui SonarLeft assuma il valore di 800, LeftBusy varrà 0.
Nella dicharazione dei predicati è possibile anche usufruire delle operazioni

booleane NOT, AND e OR. Un esempio è il seguente:
IRDirClose = (OR (P IRDirCloseR) (P IRDirCloseL));
In logica fuzzy le operazioni sono così definie:
(NOT P) = 1 - val(P)
(AND P T) = min(val(P), val(T))
(OR P T) = max(val(P), val(T))
dove P e T rappresentano due predicati e la funzione val ritorna il valore del predicato.

2.7.3 Scelta dei comportamenti da attivare

Avendo calcolato tutti i predicati è giunta l'ora della valutazione dei comportamenti. Mr. Brian permette di scegliere i comportamenti da attivare. Infatti, in un certo istante potrebbe accadere che non tutti i comportamenti debbano essere attivi. Questa scelta viene fatta attraverso le regole che Mr. Brian definisce CANDO. Un esempio di un robot calciatore:
$CalciaPalla => (PPallaInPossesso)$
in queto caso, come è facilmente intuibile, il comportamento CalciaPalla viene attivato solamente nel caso in cui la palla sia in possesso del robot.

2.7.4 Valutazione delle regole

Giudicati i comportamenti attivi, è ora di considerare le regole che compongono ogni comportamento.
Ogni regola è fatta nel seguente modo:
$(predicatoIn) => (varOut1valore1)(varOut2valore2)\ldots$
predicatoIn è un certo predicato che avrà un certo valore di verità. PredicatoIn potrebbe essere anche formato da più predicati uniti fra loro tramite gli operatori booleani elencati precedentemente. VarOut1, varOut2... sono invece le variabili di uscita con valore1, valore2... i valori che tali variabili devono assumere.

2.7.5 Inferenza

Calcolato il valore di verità del predicato d'ingresso, bisogna ora attivare l'insieme fuzzy d'uscita.
Questo processo in cui si passa dall'antecedente della regola al conseuente prende il nome di inferenza.

Nei casi particolari in cui predicatoIn assuma il valore di 0 o 1, anche i valori di verità delle variabili d'uscita assumeranno rispettivamente i valori 0 o 1. Negli altri casi, tali uscite, assumeranno valori intermedi dipendenti dal predicato in ingresso.

2.7.6 Preferenza

Nelle regole, sia di comportamenti differenti che dello stesso comportamento, può essere presente più volte la stessa variabile di uscita e quindi, al variare del valore di verità del predicato che la precede e dell'insieme fuzzy di appartenenza, può assumere valori di crisp diversi. A questo punto si associano ad ogni coppia variabile-valore un'importanza diversa (che Mr. Brian definisce attraverso regole di tipo WANT). Il valore delle variabili di uscita sarà quindi la media, pesata sul valore di importanza, dei valori che assumono.

2.7.7 Defuzzificazione

Defuzzificazione è il processo che trasforma le azioni, con i relativi valori di verità, in dati finali. Per far ciò bisogna anche specificare le funzioni che descrivo le variabili di uscita. Mr. Brian mette però a disposizione solo funzioni di tipo singleton. Un esempio di funzione di uscita è il seguente:
(TANSPEED
(SNG (VERY_FAST_FORWARD -80))
(SNG (FAST_FORWARD -50))
(SNG (FORWARD -35))
(SNG (SLOW_FORWARD -20))
(SNG (VERY_SLOW_FORWARD -10))
(SNG (STEADY 0))
(SNG (VERY_SLOW_BACKWARD 10))
(SNG (SLOW_BACKWARD 20))
(SNG (BACKWARD 35))
(SNG (FAST_BACKWARD 50))
(SNG (VERY_FAST_BACKWARD 80))
)
Il processo di defuzzificazione consiste nel calcolare la media dei vari valori associati ad una variabile pesata sull'importanza di ogni coppia variabile-valore.

2.7.8 Valutazione di un nuovo livello di comportamenti

Mr. Brian permette di dividere i comportamenti per livelli. Si parte ad eseguire i comportamenti dei livelli più bassi sino ad arrivare a quelli più alti. Ogni livello può accedere alle azioni proposte dai livelli precedenti e può anche cancellarle.
La figura 2.12 schematizza questo fatto.

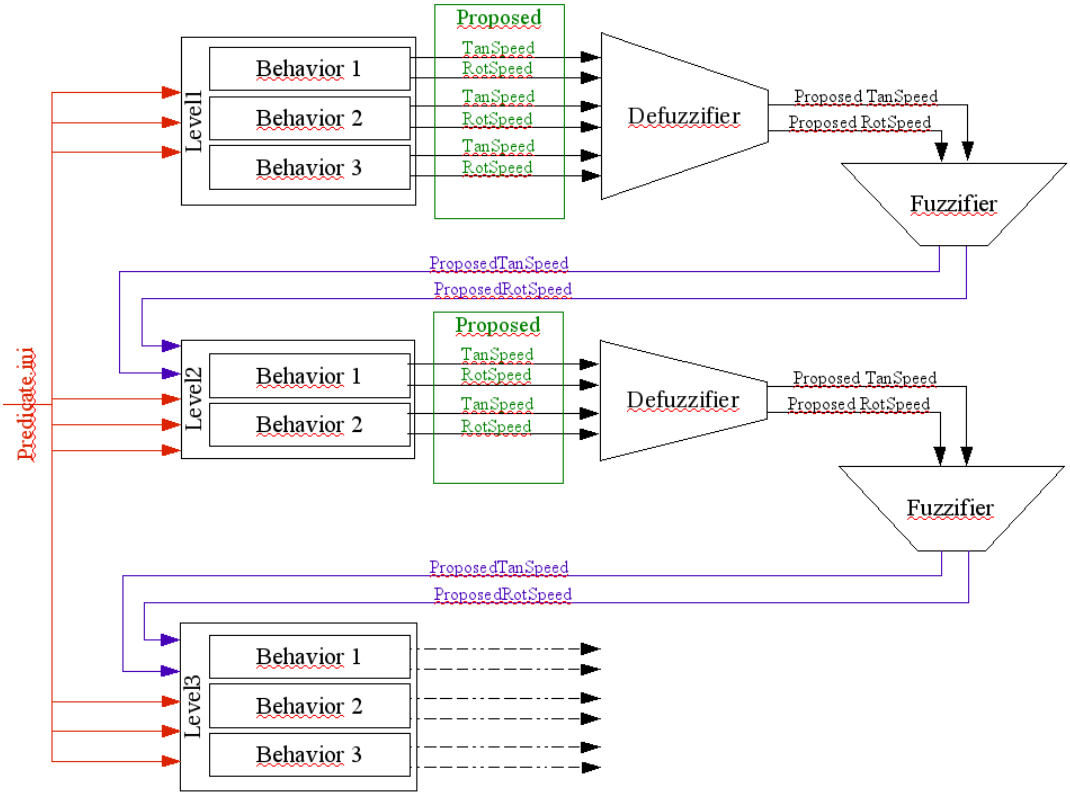

Figura 2.12: Flusso delle azioni proposte.

2.8 DCDT

Per avere una struttura più flessibile in modo da poter sperimentare diverse soluzioni e senza per questo dover riprogettare tutta l'architettura, si è diviso il progetto in diversi moduli. Infatti un robot può essere dotato di più componenti, come mostrato in figura 2.13.

Per far comunicare i vari moduli si è utilizzato un programma che ne gestisce la comunicazione: DCDT (Device Communities Developement Toolkit) [2]. Questo, sviluppato nell'ambito del progetto MRT, è in grado di far comunicare in modo semplice i vari moduli tramite l'invio di messaggi.

DCDT è un architettura multithread formata da un oggetto principale chiamato Agorà e da diversi Membri (uno per ogni modulo). I vari membri vengono eseguiti o periodicamente o per la notifica di qualche evento.

Per lo scambio dei messiggi i vari Membri usano una politica di pubblicazioni e sottoscrizioni con messaggi appartenenti a tipi diversi. In questo modo, ogni Membro che vuole ricevere un certo tipo di messaggio lo comunica al gestore dei messaggi (Dispacher). Quando poi un Membro vuole mandare un messaggio di un certo tipo lo passa al Dispacher che provvede a smistarlo a chi ne aveva fatto richiesta.

Ecco di seguito un esempio di come creare il Dispacher e i vari Membri.

```
// Create the Dispatcher
ModuleDispatch* Dispatch = new ModuleDispatch ("", basename(argv[0]));

// Create the Agent list
std::vector < StringModuleMember * >agents;

#ifdef SONAR
  \\Create Sonar Member
  SonarExpert* pSonar = new SonarExpert (Dispatch, sonar_period);
  \\Add Sonar to Member list
  agents.push_back(pSonar);
#endif

\\Addition the Members to Dispacher
for (std::vector < StringModuleMember * >::iterator i = agents.begin ();
    i != agents.end (); i++)
  Dispatch->AddMember (*i);

\\Activation of the Members
for (std::vector < StringModuleMember * >::iterator i = agents.begin ();
```

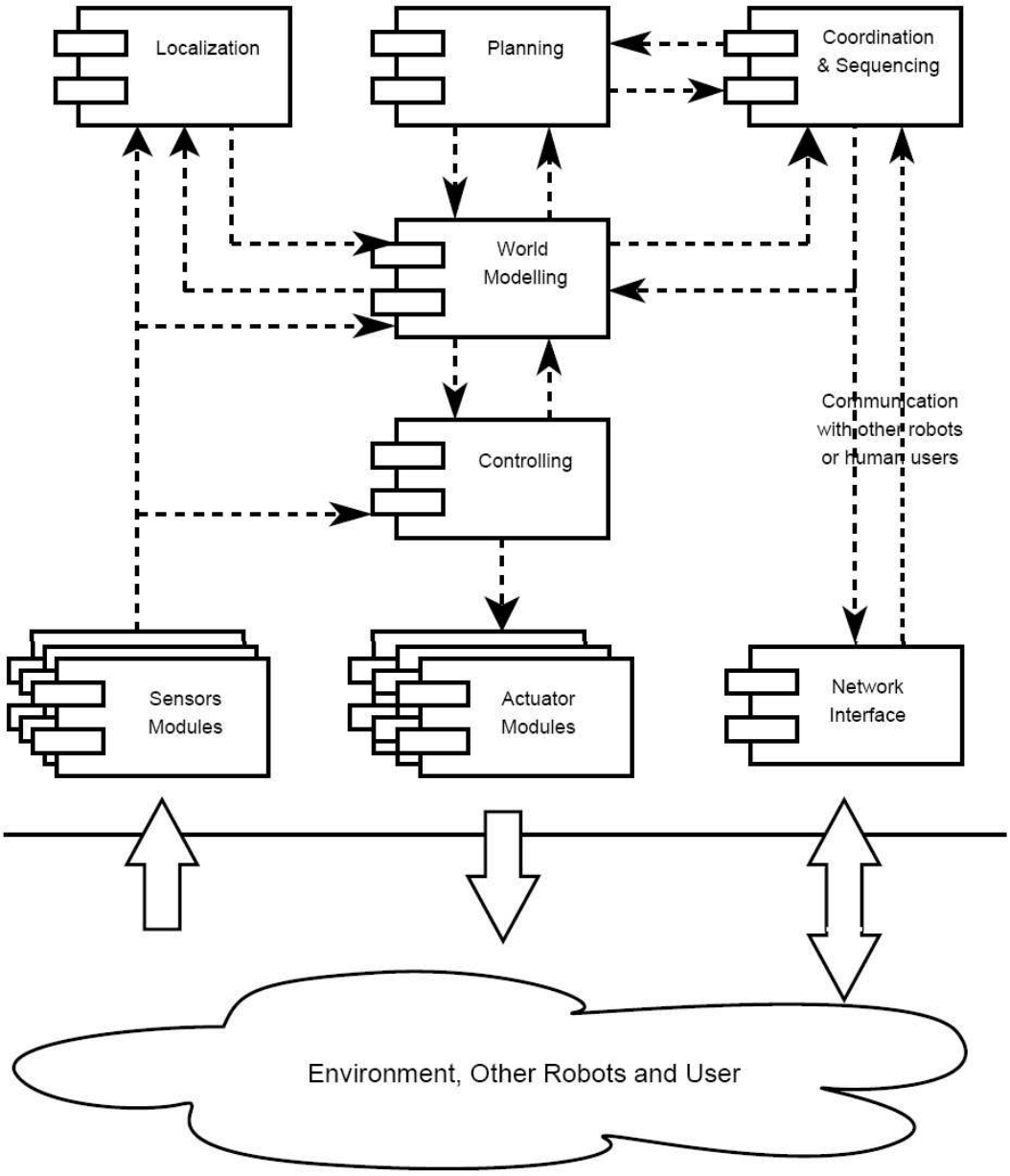

Figura 2.13: Generale suddivisione in moduli

```
        i != agents.end (); i++)
      Dispatch->ActivateMember (*i);

  \\Activation of the Dispacher
  Dispatch->LetsWork ();
```

Ogni Membro dovrà ereditare dalla classe StringModule Member e implementare i metodi: RunInit(), RunDuty() e RunClose() in modo che l'Agorà richiami i vari metodi rispettivamente: alla creazione del Membro, ad ogni determinato periodo di tempo e alla cancellazione del Membro.

Capitolo 3

RoboWii 2.0

3.1 Il programma

Per realizzare il gioco si è deciso di modificare la precedente implementazione (RoboWii 1.0). Questa, era stata fatta modificando il programma che gestisce i robot calciatori all'interno del progetto MRT. Scritto in C++, è formato da diversi moduli comunicanti tra di loro attraverso DCDT (cap. 2.8). Tra i moduli compare anche Mr. Brian (cap. 2.7) che gestisce i comportamenti scritti in logica fuzzy.
Nel corso di questo progetto, è stato aggiunto il modulo per la gestione di Spykee e il modulo per la gestione dei sonar, delle notevoli modifiche sono state fatte al gestore delle regole di gioco.
Dato che il robot ne era sprovvisto, è stato eliminato il modulo che gestiva l'odometria, ed è stato sostituito con un modulo che, in base ai messaggi destinati ai motori, calcola il modello cinematico.

I messaggi inviati tra i vari moduli sono mostrati in figura 3.1. Ecco una breve spiegazione dei loro contenuti dei messagi:
- MSG_TO_LOG contengono le posizioni dei led infrarossi visti dal Wiimote.
- MSG_POSITION contengono la posizione del robot nell'ambiente di gioco.
- MSG_FROM_MOTION contengono le azioni eseguite dal robot.
- MSG_TO_MOTION contengono i movimenti da far eseguire al robot.
- MSG_TO_BRIAN contengono le variabili fuzzy da analizzare attraverso i comportamenti.
- MSG_TO_WIIM contengono le distanze percepite dai sonar.
- MSG_TO_SONAR contengono il livello di carica.

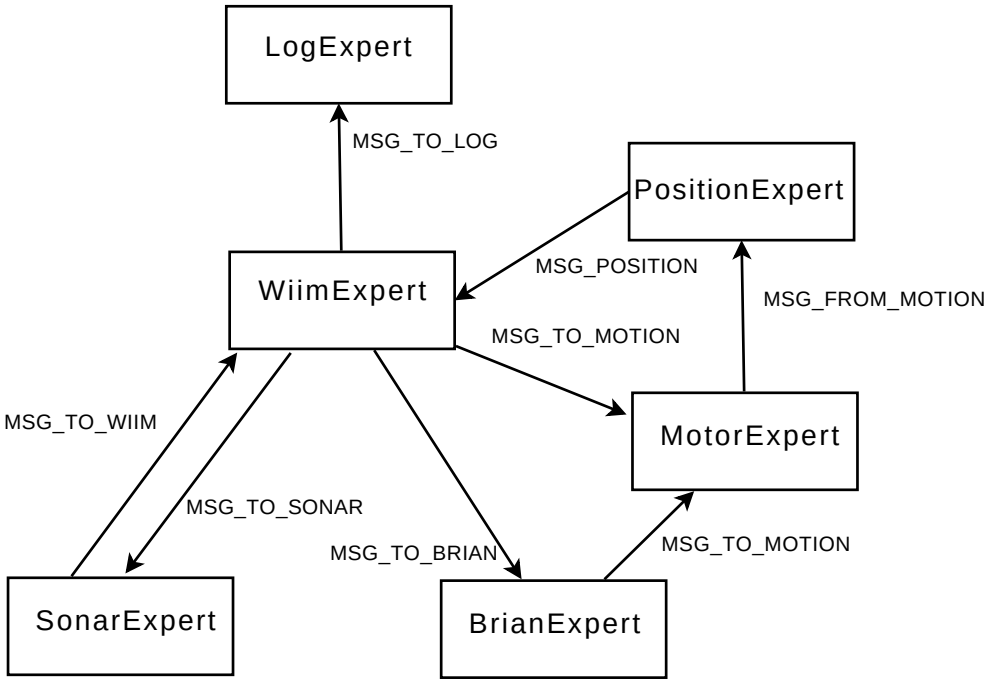

Figura 3.1: Schema dei messaggi inviati tra i vari moduli.

3.1.1 Descrizione dei messaggi

MSG_TO_LOG, indica al modulo che si occupa del log, se il wiimote sta puntando il robot e la posizione del puntamento rispetto al centro delle spalle del robot. Questo messaggio è formato nel seguente modo:

```
valid VAL xcenter CENTERX ycenter CENTERY
```

in cui VAL vale 1 se il robot è puntato e 0 se invece non lo è. CENTERX e CENTERY indicano rispettivamente le coordinate x e y del puntamento.

MSG_FROM_MOTION sono messaggi inviati dal gestore dei motori al modulo che si occupa di calcolare la posizione nell'ambiente. Sono formati nel seguente modo:

```
<D> TanSpeed VALORE </D>
<D> RotSpeed VALORE </D>
```

I messaggi del tipo MSG_TO_MOTION vengono invece ricevuti dal gestore dei motori e sono così formati:

```
<D> movement VALORE </D>
<D> orientation VALORE </D>
```

I messaggi del tipo MSG_TO_BRIAN vengono inviati a Mr. Brian e sono formati da insiemi di:

```
<D> PREDICATO VALORE </D>
```

in cui PREDICATO sono i nomi dei predicati in ingresso e VALORE è il valore di crisp che assumono.

Il modulo che gestisce i sonar li rappresenta tramite punti cardinali. I messaggi di tipo MSG_TO_WIIM, che contengono le distanze dei sonar, sono formate dal punto cardinale e dalla distanza lungo quella direzione:

```
North dist1 NE dist2 East dist3 SE dist4 South dist5
SW dist6 West dist7 NW dist8
```

I messaggi MSG_TO_SONAR contendono il livello di carica da far visualizzare alla scheda sonar. Questi messaggi sono formati da:

```
Charge VALC Point VALP
```

in cui VALC rappresenta il valore della carica e VALP vale 1 se il robot è sotto puntamento e 0 altrimenti.

3.2 Analisi di Spykee

Dopo aver montato Spykee, la Meccano mette a disposizione solo il proprio programma esclusivo per la gestione del robot (figura 3.2).

Figura 3.2: Schermata del programma gestore di Spykee.

Questo programma non è idoneo per il nostro scopo, in quanto comandabile solo dall'utente; è necessario, invece, riuscire a gestire il robot da un programma C++. Affinché ciò avvenga, bisogna comprendere come Spykee e il suo programma di gestione comunichino tra di loro.
Innanzi tutto, Spykee dispone di una scheda di rete wireless e crea una rete wireless ad hoc, che permette di inserirsi per poi comandarlo. Connettendoci a questa rete si sentiranno tre bip provenienti dall'altoparlante di Spykee che conferma la connessione. Adesso è possibile utilizzare il programma per iniziarne l'utilizzo.
Usando un programma di sniffing si è riusciti ad analizzare la comunicazione tra Spykee e il proprio programma di gestione, in modo da poter comandare il robot dal programma scritto da noi. L'Appendice A elenca con precisione i passaggi effettuati e analizza i messaggi scambiati.

Dai messaggi, si può ad esempio desumere come comandare i motori del robot. Le velocità dei cingoli non sono fisse ma possono variare da 0 a poco meno di 0,5 m/s. Il grafico di figura 3.3 mostra il rapporto che c'è tra la velocità comandata tramite messaggio e la velocità in m/s delle ruote.
Tramite i messaggi è possibile impostare, separatamente, le velocità di og-

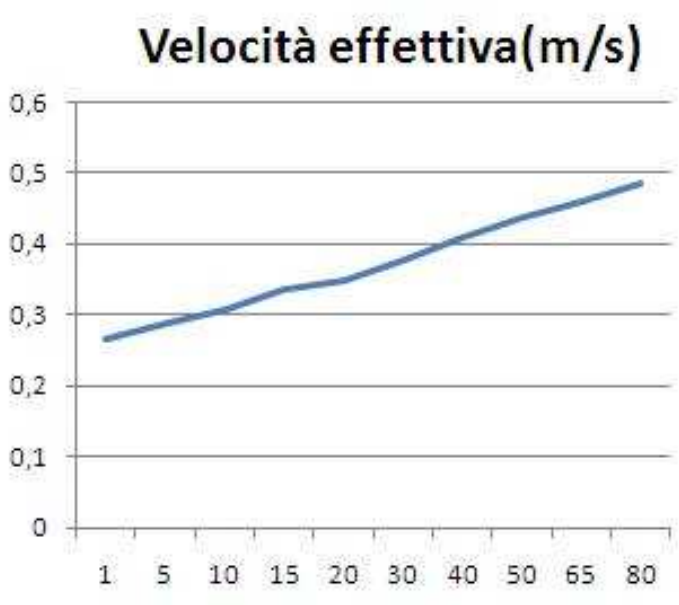

Figura 3.3: Grafico velocità in m al secondo / velocità inviata.

ni cingolo. E' inteso che per far curvare il robot bisogna impostare velocità diverse ai cingoli. Più la differenza di velocità è ampia, più strette saranno le curve effettuate. Nel caso in cui le velocità siano opposte, il robot gira su se stesso.
Tramite altri messaggi è possibile anche ricevere le immagini con una frequenza di circa 16 fotogrammi al secondo. Le immagini sono ricevute da Spykee secondo lo standard JFIF.
Una funzione particolare del robot gli permette di agganciarsi automaticamente alla base. Nell'usare questa funzione bisogna avere qualche accortezza, infatti se viene dato il comando di collegarsi mentre tra il robot e il caricabatterie è interposto un qualsiasi oggetto, Spykee non vedrà mai la base e continuerà a girare su se stesso. A volte capita che, pur vedendo il caricabatterie, non si allinei perfettamente, mancando così l'aggancio.

3.3 La classe Spykee

Per rendere modulare il programma, è stata creata una classe che si occupi di connettersi a Spykee e a tutta la gestione. I metodi che offre la classe sono i seguenti:
- Spykee(char* u, char* p), costruttore della classe (u è una stringa contente il nome utente per la connessione col robot, p contiene la password per la connessione)
- Spykee(), distruttore della classe
- void unplug(), permette di staccarlo dal caricabatterie
- void setplug(), permette di attaccarlo al caricabatterie
- void move(int l, int r), muove il robot (l è la velocità del cingolo di sinistra, r è la velocità del cingolo di destra)
- void startCamera(), inizializza l'arrivo di immagini dalla telecamera
- int getState(), restituisce lo stato della connessione
- void soundAllarm(), riproduce un suono tipo allarme
- void soundBomb(), riproduce un suono tipo bombe
- void turnOnCameraLight(), accende la luce sotto la telecamera
- void turnOffCameraLight(), spegne la luce sotto la telecamera
- static unsigned char* parseMessage(unsigned char* m, int* lm, unsigned char ** image, enum operations op), metodo statico gestore dei pacchetti in arrivo

Rispetto all'ultimo metodo c'è da fare qualche precisazione. Esso viene eseguito in un thread a parte rispetto alla classe Spykee. Dai pacchetti che riceve, il metodo estrapola solo la carica della batteria e le immagini, salvandole e passandole alle opportune richieste. Il metodo salva tutte le immagini che arrivano dal robot archiviandone un massimo di due e sovrascrivendo l'ultima alla più vecchia. Così facendo, in qualsiasi momento arrivi una richiesta per un'immagine, il metodo ne restituisce subito una. Quando arriva questo tipo di richiesta, per rendere tutto più veloce, il metodo non copia tutta l'immagine, ma restituisce il puntatore all'immagine che già ha. In questo modo il metodo non può più modificare quell'immagine perché altrimenti si rischierebbe di modificarla mentre è in uso da chi l'aveva richiesta. Per questo, dopo aver effettuato una richiesta per un'immagine ed averla usata per i propri scopi, è necessario comunicare a parseMessage che l'immagine è disponibile per eventuali modifiche.
Per le richieste viene usato il tipo di enumeratore operations:

```
enum operations
{
  parseNewMessage,
  takeChargeState,
  takeCameraPictureWithLock,
  unlockPicture
}
```

parseNewMessage viene usato per indicare a parseMessage(unsigned char* m, int* lm, unsigned char ** image, enum operations op) che c'è un nuovo messaggio da analizzare. Per questo utilizzo gli argomenti da passare sono i seguenti: in m c'è da mettere il puntatore al messaggio, in lm il puntatore ad una variabile contenente la lunghezza del messaggio, in image NULL e in op bisogna passare parseNewMessage. Ciò che restituisce il metodo è NULL.
takeChargeState viene usato per acquisire il livello di carica della batteria. Questo livello viene rilevato dalla funzione tramite l'analisi dei messaggi che indicano lo stato del robot. Per richiamare il metodo allo scopo di questo utilizzo bisogna porre m a NULL, lm deve essere un puntatore ad un intero dentro il quale si troverà poi il livello di carica, image è da mettere a NULL e op è da porre uguale a takeChargeState. Il metodo restituirà NULL.
takeCameraPictureWithLock serve per acquisire una nuova immagine, sempre ricordandosi di sbloccarla appena finito di usarla. In questo caso il metodo è da richiamare passando m pari a NULL, in lm il puntatore ad un intero in cui poi ci si aspetti di avere la lunghezza dell'immagine, in image bisognerà passare un puntatore ad un array di usigned char e alla fine image conterrà il puntatore ad un puntatore che a sua volta punterà all'immagine e infine porre op pari a takeCameraPictureWithLock. Il valore ritornato dalla funzione sarà il puntatore all'immagine.
unlockPicture indica di sbloccare l'immagine che si è acquisita in precedenza. Tutti i parametri di parseMessage sono nulli tranne op che dovrà acquisire il valore unlockPicture. Il valore restituito dalla funzione sarà NULL.
Per il movimento, la classe Spykee verrà richiamata dal gestore dei motori. A questo modulo le velocità arrivano come velocità tangenziale e velocità rotazionale ma la nostra classe le richiede come velocità del cingolo destro e del cingolo sinistro. Per ottenerle dalle precedenti bisogna applicare le seguenti formule:

$$dxSpeed = tanSpeed - rotSpeed \tag{3.1}$$

$$sxSpeed = tanSpeed + rotSpeed \tag{3.2}$$

dove dxSpeed e sxSpeed sono le velocità dei due cingoli e tanSpeed e rotSpeed sono rispettivamente le velocità tangenziale e rotazionale.

3.4 Il puntamento

Il Wiimote, tramite la telecamera, permette di riconoscere fino a quattro led infrarossi. Mettendo quindi due led infrarossi sulle spalle di Spykee saremo in grado di riconoscere quando lo si punta. Il Wiimote e la libreria Wiiuse semplificano molto l'utilizzo di questa tecnologia fornendo allo sviluppatore le coordinate degli emettitori infrarossi sul sensore della videocamera. Avendo quindi le coordinate dei due punti è possibile vedere se il Wiimote è puntato tra le spalle del robot o all'esterno. Anche se il Wiimote fosse puntato fuori dal cenro dei led, esso riuscirebbe ad accorgersi che il robot è quasi puntato. Questo sarà utile per far iniziare la fuga al robot.

3.4.1 Il circuito led

Sulle spalle sono stati montati due led Vishay TSLA6400 ad una distanza di circa trenta centimetri (figura 3.4).

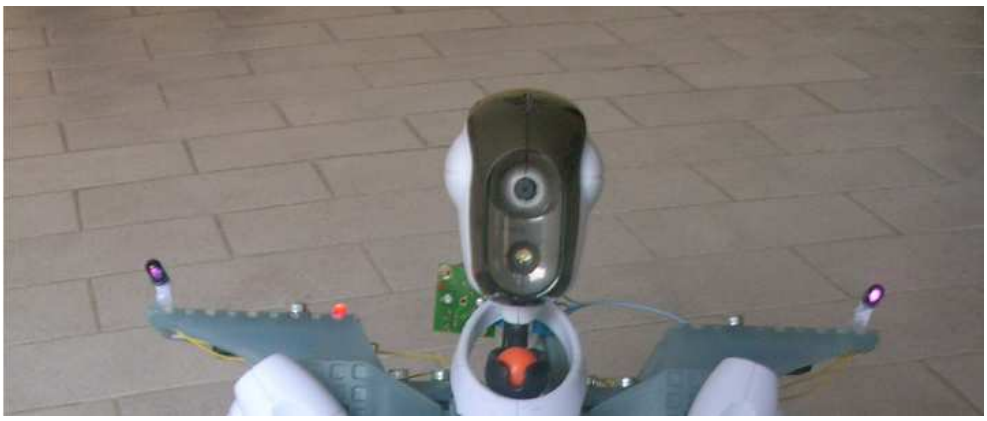

Figura 3.4: I led infrarossi montati sulle spalle del robot.

Il nostro circuito per l'accensione dei led deve ottenere l'alimentazione da tre batterie di tipo AA poste in serie. Dato che una batteria riesce a mantenere una tensione di 1.2V, ai capi del nostro circuito ci sarà una tensione pari a 3.6V. Dai datasheet dei led leggiamo che per avere la massima irradiazione deve scorrere nei led una corrente di 100mA e che con questa

corrente, la tensione ai capi di un led è pari a 1.35V. Per ridurre al minimo il circuito si potrebbe pensare di mettere i due led in serie. Così facendo, però, bisognerebbe avere ai capi dei due una tensione di 2.70V che, essendo minore di 3.6V, riusciamo a fornire. In serie ai led andrà anche una resistenza per tenere il valore della corrente costante a 100mA. Il valore della resistenza viene calcolato nel seguente modo:
$V\_tot = 3.6\ V$
$V\_led = 1.35\ V$
$V\_resistenza = V\_tot - V\_led1 - V\_led2 = 0.9V$
$I\_led = 100\ mA$

$$R = \frac{0.9V}{0.1A} = 9\Omega \tag{3.3}$$

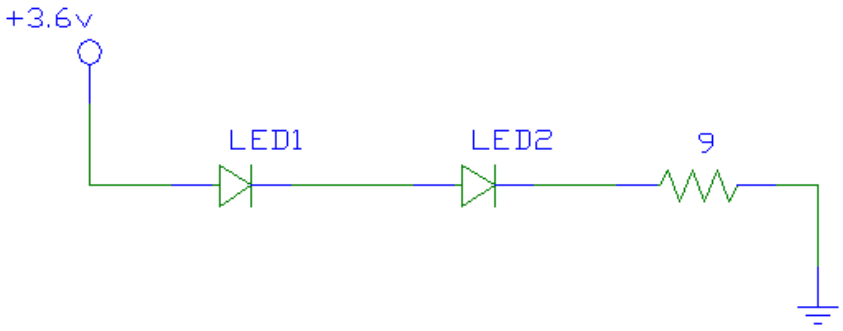

Figura 3.5: Schema circuitale.

3.5 Sonar

Dell'ambiente in cui si svolge il gioco non è disponibile una mappa perché potrebbe cambiare tra un istante di gioco ed un altro (ad esempio possono essere presenti persone in movimento o è possibile che qualche oggetto venga spostato) e comunque non è possibile pensare che una persona, prima di mettersi a giocare, crei una mappa della stanza di gioco. Quindi si sono introdotti i sonar per conoscere la posizione degli ostacoli e far in modo di ricalcolare la traietteria runtime attraverso i dati delle distanze ricevute.

Come anticipato, i sonar sono utilizzati tramite un'apposita scheda che si occupa di leggere le distanze dai sonar e di mandarle al pc tramite la porta seriale. A dire il vero la scheda sonar, montata sul robot, e il pc, appoggiato invece su una scrivania, possono essere abbastanza distanti e quindi, tra i due, non possono esserci cavi che li collegano. Per questo si è deciso di collegare la scheda sonar ad un modulo XBee che, a basso costo, assicura una buona affidabilità del collegamento e una buona velocità.
Sono stati montati in totale sei sonar: tre dietro, uno davanti, uno a destra e uno a sinistra. Le letture dei sonar possono a volte interferirsi. Questo caso capita quando si vogliano utilizzare contemporaneamente sonar vicini. Inviando assieme l'onda sonora, potrebbe capitare che quella di un sonar rimbalzi e raggiunga l'altro prima di essere raggiunto dalla propria l'onda. Questo inconveniente è stato risolto attivando due sonar alla volta che guardano in direzioni opposte.

3.5.1 La classe gestore dei sonar

La classe che gestisce la scheda sonar è SonarExpert. Questa si collega alla scheda sonar e acquisisce le distanze lette. Questi dati vengono poi inviati tramite i messaggi di tipo MSG_TO_WIIM al modulo che gestisce il gioco. Questa classe è predisposta per accettare otto sonar che identifica tramite punti cardinali: nord, nordest, est, sudest, sud, sudovest, ovest e nordovest. Un'ulteriore funzione di questa classe è quella di acquisire il livello di carica di puntamento, tramite i messaggi, MSG_TO_SONAR, e mandarlo alla scheda sonar per farlo visualizzare sui led.

3.6 Circuito led di visualizzazione della carica e del puntamento

Dopo aver fatto qualche prova col progetto quasi finito, si è notato che un giocatore che provava per la prima volta il robot faceva un po' fatica a capire quando lo stava puntando e quando il livello di carica era tale da permettergli di colpirlo. Quest'ultimo aspetto è sempre stato visualizzato, già in RoboWii 1.0, tramite i quattro led blu presenti sul wiimote, ma la visione di questi led risulta molto difficoltosa. Per questo si è deciso di mostrare sul robot entrambe le cose tramite dei led che risultassero ben visibili e di immediata comprensione. Quindi sono stati aggiunti quattro led rossi per indicare il livello di carica del puntamento e un led verde che, se acceso,

indica che il robot è sotto puntamento.
Giunge ora il problema di riuscire a comandare questi led. Gli unici oggetti che, fin ora, conoscono il livello di carica è il puntamento sono il wiimote e il programma che risiede sul pc. Se il primo non ha nessun modo di comunicare col robot, il secondo già lo fa per comandare i sonar. E' appunto tramite l'Xbee e la scheda sonar che potremo ricevere i dati per accendere o spegnere i vari led.
Sulla scheda sonar è montato un pic che fin ora si occupa solo di ricevere i dati dai sonar e inviarli al pc. L'idea per risolvere il problema è quella di modificare il programma sul pic in modo che riceva anche i dati per comandare i led e li trasmetta ad un circuito di comando. Purtroppo non sono disponibili cinque pin liberi in uscita al pic (cinque pari ai led da comandare), per cui non è possibile comandare direttamente i led ma i dati in uscita dovranno essere trasmessi in serie, poi salvati e interpretati così da poter comandare i led. Questa trasmissione avviene tramite i due pin del PIC RB0 e RB1, il primo utilizzato per il clock, l'altro per inviare i dati.
Per semplicità e maggior velocità, si è deciso di inviare alla scheda sonar un solo numero contente lo stato dei led. La tabella seguente mostra il valore in base ai led da accendere.

Num	Rossi accesi	Verde acceso
0	0	No
1	1	No
2	2	No
3	3	No
4	4	No
8	0	Sì
9	1	Sì
10	2	Sì
11	3	Sì
12	4	Sì

Con questa codifica si è facilitata l'interpretazione da parte del pic perché i tre bit meno significativi del numero indicano il numero di led rossi da accendere e il quarto bit indica se il led verde è acceso o meno. A questo punto il pic non fa altro che far uscire in modo seriale ciò che ha ricevuto. Sarà poi la logica esterna ad accendere i led corretti. Il circuito mostrato in figura 3.5 si occupa di ricevere i dati contenenti i led da accendere (sempre codificati come mostrato nella tabella precedente), salvarli, analizzarli e accendere i led corretti.

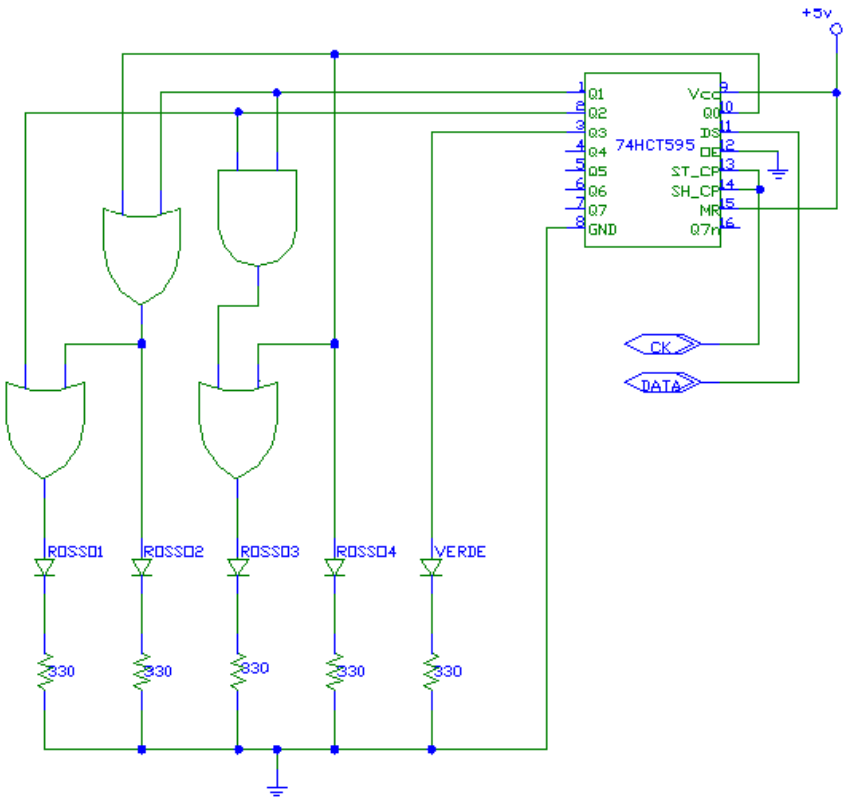

Figura 3.6: Circuito per l'accensione dei led.

I dati che escono dal pic vengono salvati in uno shift register, precisamente il 74HC595 (capace di salvare fino ad otto bit). Uno shift register non contiene altro che dei flip flop collegati in seire, che all'arrivo di un nuovo bit ognuno passa quello che memorizzava al successivo così da tener salvati solo gli ultimi bit ricevuti.
Avendo ora i dati stabili, è possibile creare il circuito logico per analizzarli. Dopo la scrittura delle tabelle di verità e la semplificazione tramite le tavole di Carnot, otteniamo quanto di seguito:
$L\_verde = Q\_3$
$L\_rosso1 = Q\_0 + Q\_1 + Q\_2$
$L\_rosso2 = Q\_0 + Q\_1$
$L\_rosso3 = Q\_0 + Q\_1 \cdot Q\_2$
$L\_rosso4 = Q\_0$
dove $L\_nomeled$ indica il segnale che pilota un certo led. La resistenza in serie ai led è stata tarata in modo da far passare 15mA:

$$R = \frac{5V}{0.015A} \approx 330\Omega \tag{3.4}$$

Per realizzare il circuito sono stati quindi utilizzati:
- 5 resistenze da 330Ω
- un 74HC595 (shift register)
- un 74HC08 (quattro porte AND a due ingressi)
- un 74HC32 (quattro porte OR a due ingressi)

la scelta della famiglia logica è caduta sulla HC perché, anche se più lente della LS, garantisce il funzionamento da 2V a 6V. Nel nostro caso, le batterie non riescono a garantire un tensione sempre costante, ma questa andrà via via abbassandosi proporzionalmente alla carica. Quindi un più ampio range di funzionamento ci garantisce un più elevato tempo di funzionamento.

Capitolo 4

Gioco

4.1 Descrizione del gioco

Come accennato brevemente nel primo capitolo l'utente tiene in mano il Wiimote e con questo deve puntare il robot per almeno quattro secondi sulle spalle. Nel frattempo il robot fa di tutto per sottrarsi all'avversario.

Ogni volta che il giocatore colpisce Spykee gli viene assegnato un punto; se, viceversa, nell'arco di un minuto non vi riesce, il punto viene assegnato al robot. Quando il punto viene assegnato al robot questo balla ed emette suoni, se invece viene assegnato al giocatore emette un rumore tuonante. Il gioco termina quando uno dei due totalizza tre punti, vincendo la gara.

Figura 4.1: Led indicativi sul robot.

Per rendere più arduo il gioco è stato introdotto un livello di carica che va da zero a quattro. Quando il robot è puntato il livello si alza di un'unità ogni secondo, quando il robot esce dal puntamento il livello tende a scendere con la stessa frequenza. Per poter colpire il robot il livello di carica deve essere pari a quattro, altrimenti tutta la carica raggiunta fino a quel momento viene persa. Il livello di carica è riscontrabile sia attraverso i led presenti sul Wiimote che attraverso i led rossi (figura 4.1) posizionati sul robot. Su

Figura 4.2: Prova di gioco.

quest'ultimo è presente anche un led verde che, quando acceso, indica che il robot si trova sotto mira.
Tasti del Wiimote:
- per sparare usare il tasto B
- in caso di emergenza, il tasto - (meno) ferma il gioco

4.2 Comportamenti e configurazione di Mr. Brian

Una volta che tutti i sensori e tutti gli attuatori sono stati configurati correttamente e in modo da essere pronti all'uso, si presenta il problema di comandare gli attuatori in base ai dati sensoriali in ingresso.
Per far ciò è stata utilizzata un'architettura simile a quella subsumption in cui i comportamenti corrispondono a delle regole fuzzy. La differenza con l'architettura subsumption è che possono essere eseguiti più regole per volta e calcorare la media ponderata per le uscite. Per realizzare tale architettura è stato utilizzato Mr. Brian.
Per la realizzazione del gioco sono stati scritti sei comportamenti:

- Snaking, verifica, tramite gli accellerometri del wiimote, che si stà tentando di colpire il robot facendogli iniziare la fuga
- EscapeIR, si occupa di far allontanare il robot dal puntamento del wiimote
- BurstSpeed, permette al robot di girarsi e scappare in avanti
- RobotWin, fa ballare il robot quando vince
- TurnObstacles, evita al robot di impattarsi contro gli ostacoli
- TraiettoryCorrection, corregge la traiettoria del robot facendolo allontanare dagli ostacoli che potrebbero dar fastidio in seguito.
  I comportamenti sono stati elencati da quelli appartenenti ad un livello minore a quelli appartenenti ad un livello maggiore. Come descritto nel cap. 2.7.7 i comportamenti ad un livello maggiore possono conosce e modificare le azioni proposte dai comportamenti di livello minore.

Gli ingressi di questi comportamenti sono i seguenti:

- IRCenter, indica la distanza tra il punto puntato dal wiimote e il centro del robot

- Hit, vale 1 quando l'utente fa un punto, -1 quando il robot fa un punto, 0 altrimenti
- Clock, clock con periodo di due secondi
- GAcc, contiene la somma delle accellerazioni subite dal wiimote
- WiiMotePitch, indica l'inclinazione del wiimote
- Random, variabile casuale contenente numeri da 0 a 5
- GoForward, se vera indica di scattare diritto
- SonarLeft, distanza sul sonar di sinistra
- SonarRight, distanza sul sonar di destra
- SonarBack, distanza sul sonar posteriore
- SonarFront, distanza sul sonar davanti
- SonarSouthWest, distanza sul sonar dietro e spostato verso sinistra
- SonarSouthEast, distanza sul sonar dietro e spostato verso destra
- RightLeftDifference, differenza tra la distanza di destra e quella di sinistra
- ProposedTanSpeed, velcità tangenziale proposta dai livelli precedenti

e le uscite che vanno a comandare i motori del robot:

- TanSpeed, velocità tangenziale
- RotSpeed, velocità rotazionale

Gli insiemi fuzzy per le variabili di ingresso e uscita si trovano nell'appendice B e non sono descritti perché di facile comprensione.

4.2.1 Snaking

Questo comportamento fa mettere in fuga il robot quando il giocatore si appresta a mirarlo. Quando il giocatore vuole iniziare a mirare il robot, inclina la punta del wiimote verso il basso. Se inclinandolo lo ruota anche verso sinistra, il robot si troverà a sinistra del giocatore e, quindi, per iniziare ad evitare il puntamento il robot dovrebbe ruotare verso destra. Viceversa, se il wiimote viene inclinato e ruotato verso destra, Spykee ruoterà, invece,

verso sinistra.
Sono state poi aggiunte anche delle rotazioni casuali che avvengono in media ogni cinque secondi e durano un secondo.
Le regole fuzzy di questo comportamento sono:
$(AND(Pointing)(GAccWeakR)) => (RotSpeedRIGHT)$;
$(AND(Pointing)(GAccStrongR)) => (RotSpeedFAST\_RIGHT)$;
$(AND(Pointing)(GAccWeakL)) => (RotSpeedLEFT)$;
$(AND(Pointing)(GAccStrongL)) => (RotSpeedFAST\_LEFT)$;

$(TurnRandomLeft) => (RotSpeedFAST\_LEFT)$;
$(TurnRandomRight) => (RotSpeedFAST\_RIGHT)$;
In cui la definizione dei predicati di sinistra è mostrata nell'appendice A.
In questo comportamento, sarebbe possibile aggiungere anche una regola per farlo continuare ad andare in avanti, in questo modo il robot andrebbe in giro per il locale, anche se non è puntato.

4.2.2 EscapeIR

Questo comportamento si occupa di far scappare il robot quando è sotto puntamento.
Se Spykee è puntato, tende ad andare indietro e a ruotare nella direzione opposta verso cui è puntato. Ossia, dato che tenta di allontanarsi in retromarcia, se il wiimote è puntato nalla parte destra del robot, esso girerà verso destra e farà uguale a sinistra. Per cui le regole che compongono il comportamento sono fatte nel seguente modo:
$(IRDirClose) => (\&DEL.RotSpeedANY)(\&DEL.TanSpeedANY)$;
$(AND(NOT(OR(TurnRandomLeft)(TurnRandomRight)))(IRDirClose)) =>$
$(TanSpeedFAST\_BACKWARD)$;
$(AND(IRDirNear)(CkHigth)) => (TanSpeedBACKWARD)$;
$(AND(IRDirNear)(CkLow)) => (TanSpeedFAST\_BACKWARD)$;
$(AND(IRDirCloseR)(NOT(TurnRandomLeft))) => (RotSpeedFAST\_RIGHT)$;
$(AND(IRDirCloseL)(NOT(TurnRandomRight))) => (RotSpeedFAST\_LEFT)$;
$(AND(AND(IRDirNearR)(NOT(TurnRandomLeft)))(CkHigth)) =>$
$(RotSpeedFAST\_RIGHT)$;
$(AND(AND(IRDirNearL)(NOT(TurnRandomRight)))(CkHigth)) =>$
$(RotSpeedFAST\_LEFT)$;

4.2.3 BurstSpeed

Per rendere maggiormente difficile il gioco si è fatto in modo che, in media ogni venti secondi, il robot si giri di 180 gradi rispetto alla posizione in cui si trova e faccia uno scatto in avanti:
$(OR(GoForwardBurst)(TurnLeftBurst)) =>$
$(\&DEL.RotSpeedANY)(\&DEL.TanSpeedANY);$
$(TurnLeftBurst) => (RotSpeedFAST\_LEFT);$
$(AND(GoForwardBurst)(NOT(OR(IRDirClose)(IRDirNear)))) =>$
$(TanSpeedFAST\_FORWARD);$
Se, durante lo scatto in avanti, dovesse trovarsi di fronte il giocatore che tenta di colpirlo, il robot tenta di sterzare per schivarlo:
$(AND(GoForwardBurst)(AND(OR(IRDirClose)(IRDirNear))(RightFar))) =>$

$(\&DEL.TanSpeedANY)(RotSpeedFAST\_RIGHT)(TanSpeedFORWARD);$
$(AND(GoForwardBurst)(AND(OR(IRDirClose)(IRDirNear))(LeftFar))) =>$
$(\&DEL.TanSpeedANY)(RotSpeedFAST\_LEFT)(TanSpeedFORWARD);$

4.2.4 RoboWin

Questo comportamento fa star fermo il robot nel caso in cui abbia acquisito un punto il giocatore, e lo fa ballare nel caso in cui gli vienga assegnato il punto. Il comportamento è formato da queste semplici regole:
$(OR(HumanPoint)(RobotPoint)) =>$
$(\&DEL.RotSpeedANY)(\&DEL.TanSpeedANY);$
$(HumanPoint) => (TanSpeedSTEADY);$
$(HumanPoint) => (RotSpeedAHEAD);$
$(AND(RobotPoint)(CkHigth)) => (RotSpeedFAST\_LEFT);$
$(AND(RobotPoint)(CkLow)) => (RotSpeedFAST\_RIGHT);$
Il clock varia ogni secondo, in questo modo, all'assegnazine del punto, il robot si muoverà continuamente a sinistra e a destra.

4.2.5 TurnObstacles

Questo comportamento si occupa, attraverso l'ausilio dei dati provenienti dai sonar, di far evitare gli ostacoli presenti nell'ambiente. Quando il robot ha vicino un ostacolo e le uscite dei livelli precedenti tendono a farlo andare contro, queste vengono cancellate. In più il robot viene fatto girare e allontanare dagli ostacoli in modo da potersi nuovamente muovere liberamente. Le regole del comportamento sono:

$(AND(propTanSpeedBackward)(OR(BackBusy)(OR(SouthWestBusy)$
$(SouthEastBusy)))) => (\&DEL.TanSpeedANY);$
$(AND(OR(BackBusy)(BackSemiBusy))(RightFar)) =>$
$(\&DEL.RotSpeedANY)(RotSpeedFAST\_RIGHT);$
$(AND(OR(BackBusy)(BackSemiBusy))(LeftFar)) =>$
$(\&DEL.RotSpeedANY)(RotSpeedFAST\_LEFT);$

$(FrontBusy) => (\&DEL.TanSpeedANY)(TanSpeedBACKWARD);$
$(FrontVeryNear) => (TanSpeedFAST\_BACKWARD);$
$(AND(OR(FrontBusy)(FrontSemiBusy))(RightFar)) =>$
$(\&DEL.RotSpeedANY)(RotSpeedFAST\_RIGHT);$
$(AND(OR(FrontBusy)(FrontSemiBusy))(LeftFar)) =>$
$(\&DEL.RotSpeedANY)(RotSpeedFAST\_LEFT);$
$(AND(FrontFree)(OR(BackBusy)(OR(SouthWestBusy)(SouthEastBusy)))) =>$
$(TanSpeedFORWARD);$

4.2.6 TraiettoryCorrection

Questo comportamento si occupa soltanto di far ruotare leggermente il robot in modo da anticipare il problema di evitare gli ostacoli. Nel caso il robot si trovasse a viaggiare parallelamente ad un muro, per sicurezza viene fatto allontanare. Di seguito le regole:
$(AND(propTanSpeedForward)(RightBusy)) =>$
$(\&DEL.RotSpeedANY)(RotSpeedFAST\_LEFT);$
$(AND(propTanSpeedForward)(LeftBusy)) =>$
$(\&DEL.RotSpeedANY)(RotSpeedFAST\_RIGHT);$
$(AND(propTanSpeedBackward)(RightBusy)) =>$
$(\&DEL.RotSpeedANY)(RotSpeedFAST\_RIGHT);$
$(AND(propTanSpeedBackward)(LeftBusy)) =>$
$(\&DEL.RotSpeedANY)(RotSpeedFAST\_LEFT);$

Capitolo 5

Direzioni future e conclusioni

5.1 Valutazioni conclusive

L'obiettivo finale del progetto è stato quello di creare un gioco in cui un robot autonomo interagisse con un utente umano.
Nelle prime prove, si è notato che un giocatore novello faceva fatica a capire quando stesse puntando il robot o meno. Per questo si è pensato di portare questa segnalazione sul robot assieme al livello di carica. In questo modo si riesce a capire facilmente il momento esatto per colpire il robot. Dopo un po', il gioco, non variando, risulta essere sempre uguale, diventando noioso. Ma, non avendo a disposizione un sistema di localizzazione e un robot con una buona velocità, non è stato possibile creare altri scenari di gioco e, dunque, aumentare più di tanto le possibili mosse da parte del robot. Proprio il problema della velocità si nota molto quando il robot si gira di scatto per scappare in avanti. Questa azione, che dovrebbe servire per allontanarsi il più possibile dal giocatore, fa sì che l'utente smaliziato, più veloce del robot, si ponga davanti ad esso interrompendone la fuga, così da costringerlo a prendere un'altra direzione.
Comunque è un primo risultato soddisfacente.

5.2 Sviluppi futuri

Questa sezione descrive alcuni dei possibili futuri sviluppi che potrebbe avere il gioco.
Durante lo sviluppo del progetto, abbiamo avuto delle nuove idee da poter integrare, che però non si è ancora trovato il tempo ed il modo di aggiungere. Sono comparsi anche nuovi problemi, dovuti principalmente alla com-

ponentistica originaria, che, per ora, si è solo provveduto ad aggirare senza risolverli completamente.
In questa prima fase il robot si limita alla fuga evitando di farsi colpire, al lungo andare, essendo ciò ripetitivo, giocarci diventa quindi noioso. Sarebbe interessante sviluppare nuovi livelli in cui sia il robot che il giocatore debbano portare a termine compiti più complessi. Per creare nuovi livelli si potrebbe anche far affidamento sulla telecamera frontale montata su Spykee che, pur non essendo in grado di definire bene ogni particolare, può essere usata per riconoscere oggetti monocolore e di appropriate dimensioni. Si potrebbero, ad esempio, usare delle bandierine colorate per definire una base e un punto di arrivo, o degli obiettivi da raggiungere.
La possibilità, poi, di disporre di una localizzazione precisa e con un errore fisso potrebbe essere molto utile per creare nuovi livelli di diverse difficoltà. In questo modo il robot potrebbe muoversi nell'ambiente con precisione e magari, tramite l'aiuto dei sonar, crearsi una mappa dell'ambiente.
Questi nuovi livelli si potrebbero creare anche in modo da far interagire sia più utenti che più robot. Si potrebbero formare delle squadre di robot e umani con delle precise missioni da compiere. Per far ciò, forse però non andrebbe più bene il salotto di casa ma bisognerebbe avere a disposizione delle vere e proprie arene.

Bibliografia

[1] *Brian Manual*.

[2] Marcello Restelli Andrea Bonarini, Matteo Matteuci. *MRT: Robotics Off-the-Shelf with the Modular Robotic Toolkit*. 2002.

[3] Cammarata Silvio. *Sistemi a logica fuzzy*. Etas, 1997.

[4] Wiiuse. Wiiuse, a c++ library for wiimote.

Appendice A

Analisi dei pacchetti scambiati tra Spykee e il PC

Osservando i pacchetti inviati e ricevuti dalla scheda di rete wireless del computer in cui risiede il programma di gestione, si riesce a comprendere come comandare il robot. Procedendo come descritto, si è osservato e compreso come robot e computer dialoghino.
All'avvio, il programma manda un messaggio di broadcast di tipo UDP sulla porta 9000. Il contenuto del messaggio è: 44 53 43 56 1 (questo, come i messaggi seguenti, saranno tutti espressi in esadecimale a meno che non si usino simboli alfabetici). Quando Spykee risulta collegato alla rete risponde con un messaggio di tipo UDP, sulla porta utilizzata per mandare il messaggio precedente, fatto in questo modo: 31 44 53 43 56 02 14 uid=SPYKEE seguito dal numero di serie che nel nostro caso è 0860001260. L'ultima parte del messaggio rappresenta il numero di serie del robot; lo si può leggere anche sul fondo di Spykee.
Dal messaggio ricevuto da Spykee, possiamo estrapolare l'indirizzo IP che si è assegnato, in modo che di seguito possa essere utilizzata una connessione affidabile, tramite il protocollo TCP. Infatti, è giunta l'ora di creare una connessione sulla porta 9000 del robot.
A questo punto il robot si aspetta il messaggio: 50 4B 0A 00 0C; in sua mancanza, dopo circa cinque secondi, il robot invierà un messaggio di errore e chiuderà la connessione.
Ora, il robot si aspetta l'autenticazione da parte del pc. Il messaggio per l'autenticazione è così composto: 05 seguito dal nome utente, da 05 e poi la password (di default, il nome utente è admin e la password è admin). Il nome utente e la password viaggiano in chiaro.
Se l'autenticazione ha successo il robot risponde con: 50 4B 0B 00 1E 01 10,

seguito dal suo nome e da 00 00 06 31 2e 30 2e 32 32 00 01 02.
Per concludere la fase di autenticazione, è necessario mandare al robot ulteriori due messaggi: 50 4B 12 00 00 e 50 4B 09 00 01.
Finito quanto descritto precedentemente siamo pronti a interagire con Spykee.
Circa ogni cinque secondi riceveremo un messaggio riguardante lo stato del robot: 50 4B 03 00 01 XX, dove XX rappresenta la carica della batteria nel seguente modo: se XX è minore o uguale a cento, XX identifica la percentuale di carica della batteria, se invece è maggiore di cento, indica che il robot si trova sotto carica.
Nel caso avvenga qualche errore di comunicazione giungerà al computer il messaggio 50 4B 0B 00 02 00 01.
Per comandare il movimento, bisognerà procedere nel modo seguente: per prima cosa inviare il messaggio 50 4B 05 00 02, seguito da un secondo del tipo LL RR, contenente le velocità dei cingoli. LL indica la velocità del cingolo di sinistra, RR indica invece quella del cingolo di destra. Le velocità sono espresse con valori compresi tra -80 e +80 (valori espressi in decimale): negativi per far muovere il cingolo all'indietro, positivi per farlo andare in avanti. A questo punto, è palese come comandare i movimenti del robot: per farlo curvare bisognerà, o muovere i cingoli con velocità inverse per avere delle curve strette, o far viaggiare un cingolo un po più lento dell'altro per avere curve più ampie (naturalmente il robot tenderà a curvare verso il cingolo con velocità minore). Le velocità comandate verranno attuate solo per circa un secondo, dopodiché il robot si fermerà. Questo perché potrebbe accadere che il computer si blocchi, non riesca più a mandare un messaggio con velocità nulle, e perciò i cingoli continuerebbero a girare.
Come descritto precedentemente, Spykee possiede anche una telecamera. Per acquisire le immagini dobbiamo mandare al robot due messaggi: 50 4B 0F 00 02 e poi 01 01 50 4B 0F 00 02 02 01. Successivamente inizierà l'invio delle immagini da parte di Spykee con una frequenza di circa 16 al secondo. Le immagini, essendo di qualche KB di dimensione, arriveranno divise in più pacchetti. Il primo pacchetto inizierà nel seguente modo: 50 4B 02 AA AA, dove AA AA indica la dimensione in byte dell'immagine. Tutti i successivi pacchetti, fino a che si riceveranno AA AA byte, faranno parte dell'immagine. Le immagini ricevute seguono fedelmente lo standard JPEG Interchange Format quindi, scrivendole in un file, sarà possibile aprirle con un qualsiasi programma di gestione di immagini.
Attivando la ricezione delle immagini, si attiverà automaticamente anche la ricezione del suono proveniente dal microfono posto su Skypee. Similmente alle immagini, il suono verrà diviso in più pacchetti, stavolta il primo pac-

chetto inizierà con 50 4B 01 AA AA, in cui AA AA indica la dimensione in byte del suono ricevuto.
Spykee permette anche di eseguire suoni preimpostati. A questo scopo bisogna mandare il messaggio: 50 4B 07 00 01 e successivamente bisognerà mandarne un altro contenente un numero che identificherà il tipo di suono da eseguire:
- 0 per il suono Allarme
- 1 per il suono Bombe
- 2 per il suono Lazer
- 3 per il suono Risata
- 4 per il suono Engine
- 5 per il suono Robot

Spykee permette anche di eseguire un qualsiasi altro suono, questo aspetto non è stato però analizzato non essendo utile al progetto.
C'è anche la possibilità di inviare i comandi per farlo collegare o scollegare dal caricabatterie. Per entrambi i comandi bisogna prima inviare il messaggio 50 4B 10 00 01 e poi: 05 nel caso lo si voglia staccare, 06 nel caso lo si voglia far collegare. Ad azione eseguita, il robot risponderà con 50 4B 10 00 01 01, nel caso si sia scollegato, con 50 4B 10 00 01 02 nel caso si sia agganciato.
L'attacco al caricabatterie è diviso in due fasi: il riconoscimento del caricabatterie e il moto verso di esso. Il caricabatterie è dotato di alcuni led infrarossi e così come Spykee è dotato di un rivelatore di infrarossi. La prima fase consiste nel far ruotare il robot affinché possa captare gli infrarossi e si allinei in modo da averli alle spalle. Nella seconda fase il robot procede invece in retromarcia fino a connettersi. L'automa si accorge dell'attacco tramite un pulsantino, che si trova nella parte inferiore, che nell'attaccarsi al caricabatterie viene premuto. Nell'usare questa funzione bisogna avere qualche accortezza, infatti se viene dato il comando di collegarsi mentre tra il robot e il caricabatterie è interposto un qualsiasi oggetto, Spykee non vedrà mai la base e continuerà a girare su se stesso. A volte capita che, pur vedendo il caricabatterie, non si allinei perfettamente, mancando così l'aggancio.
Per accendere o spegnere la luce posta sotto la telecamera occorre inviare il messaggio: 75 4B 04 00 02, seguito da un ulteriore messaggio, contenente 00 00 per spegnerla, 00 01 per accenderla.
Ci sono inoltre a disposizione altre funzioni, che consentono, ad esempio di eseguire i suoni desiderati o accendere delle lucine poste sul robot. Queste, non essendo utili al progetto, non sono state analizzate. Continuando col metodo seguito in precedenza sarebbe possibile analizzare e imparare ad us-

are anche queste.

Appendice B

Manuale d'uso di RoboWii 2.0

Questo manuale riporta le informazioni per una corretta esecuzione e compilazione del programma RoboWii 2.0.

B.1 Prerequisiti

Il software è stato sviluppato per computer dotati di Ubuntu 8.10 per i386 con installati i pacchetti elencati nella tabella successiva.

Nome	Versione	Descrizione
bison	1:2.3.dfsg-5	creazione parser
flex	2.5.35-2ubuntu1	creazione parser
g++	4:4.3.1-1ubuntu2	compilatore c++
libboost-program-options-dev	1.34.1-11ubuntu1	libreria gestione opzioni
libraw1394-dev	1.3.0-4	libreria per telecamera firewire
libdc1394-22-dev	2.0.2-1	libreria per telecamera firewire
libbluetooth-dev	4.12-0ubuntu5	libreria bluetooth
mesa-common-dev	7.2-1ubuntu2	libreria per finestra grafica
libsdl1.2-dev	1.2.13-2ubuntu1	libreria per finestra grafica
freeglut3-dev	2.4.0-6.1	libreria per finestra grafica
paintlib-dev	2.6.2-14ubuntu1	libreria per la manipolazione di immagini

La versione riportata è quella attualmente presente nei repository ufficiali di Ubuntu 8.10.
Il sistema operativo Ubuntu 8.10 si ritiene compreso di tutti i pacchetti di default aggiornati al ottobre 2008 (data di uscita della release).
E' necessaria poi anche l'installazione della libreria Wiiuse 0.12 (scaricabile da http://sourceforge.net/projects/wiiuse/files). Per compilarla digitare i

seguenti comandi nella cartella della libreria:
- make wiiuse
- make install (come superutente).

B.2 Esecuzione

Per eseguire il programma, recarsi nella directory robowii e lanciare il programma robowii (da consolle col comando ./robowii).
Di default il programma si aspetta che il file descrittore della porta seriale a cui è collegata la scheda sonar sia il seguente: "/dev/ttyUSB0". Nel caso il programma si accorgesse che non fosse corretto, richiederebbe, tramite il seguente messaggio "Please insert sonar name port", di inserire il percorso e il nome corretto.

B.3 Compilazione

Nel caso si volesse apportare qualche modifica al codice, prima di eseguirlo occorre ricompilarlo. Per far ciò, recarsi nella cartella robowii e lanciare lo script rw.sh (da consolle col comando ./rw.sh). Questo script compila ed esegue robowii, inoltre gestisce i file di log (per maggiori informazione leggere i commenti all'interno di esso).

B.4 Descrizione dei file importanti

I file presenti in robowii che possono risultare utili sono i seguenti:

Nome	Percorso	Descrizione
robowii	robowii/	Eseguibile principale.
checker	brian/bin/	Controlla la corretta sintassi nei file di configurazione di Mr. Brian. Passare come argomento il percorso relativo della cartella contenente i file di configurazione di Mr. Brian.
cartella brian	robowii/config/	Contiene i file di configurazione di Mr. Brian.
const.h	shared/	Varie configurazione riguardanti robowii.
kernel.cpp	robot/src/	Kernel del software, definisce e carica i moduli utilizzati.
WiimExepert.cpp	robot/src/	Expert principale: gestisce il ciclo di gioco.

B.5 Connessioni

In seguito vengono elencate le varie connessioni per l'ottima funzionalità del progetto (i numeri delle connessioni si riferiscono alle figure C.1, C.2, C.3):

1. Uscita alimentazione (non collegato)
2. Ingresso alimentazione (5V)
3. Segnale in ingresso dalla scheda sonar (filo bianco)
4. Clock (filo blu)
5. Connessione led rossi
6. Connessione led verde
7. Connessione led gialli
8. Connettore +5V
9. Connettore massa
10. Interruttore accensione
11. Alimentazione scheda sonar (5V)
12. Connessione seriale (da collegare all'XBee)
13. Connessione led per l'indicazione della trasmissione/ricezione (opzionale)

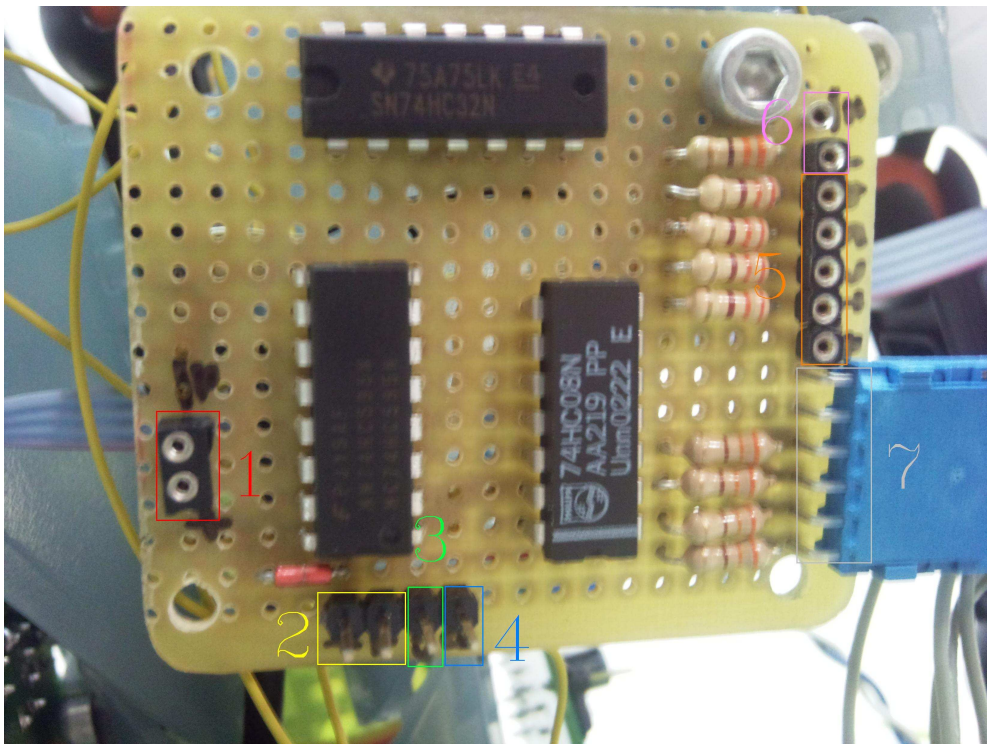

Figura B.1: Circuito per l'accensione dei led.

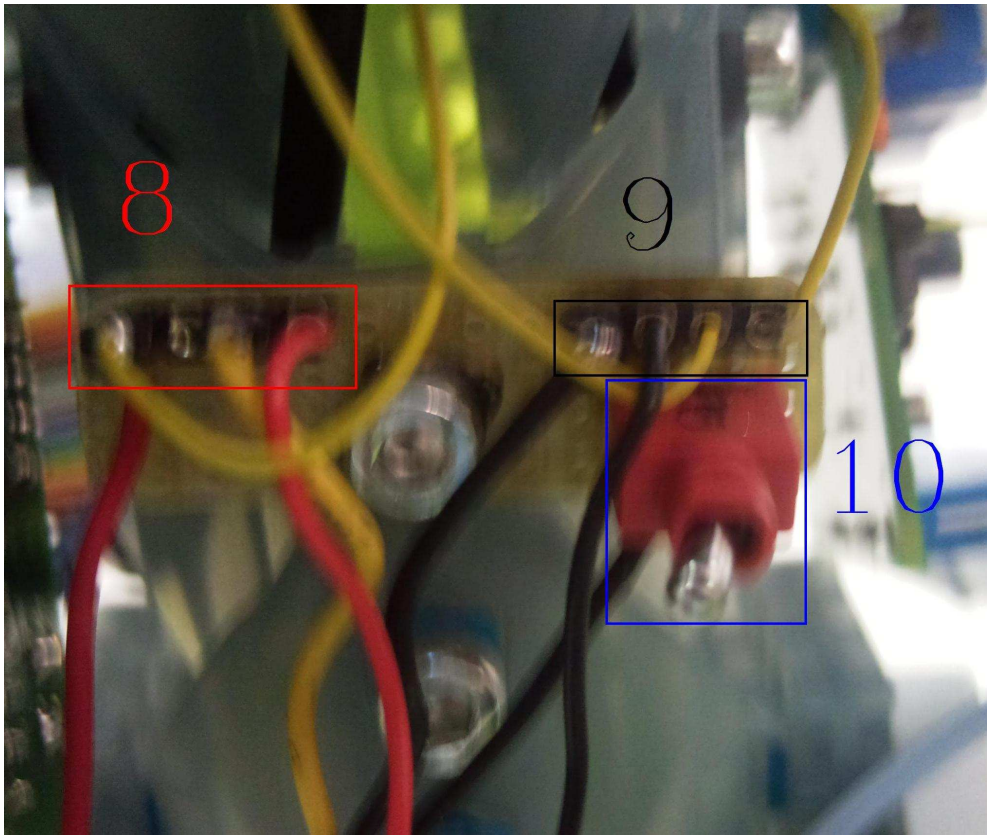

Figura B.2: Alimentazione.

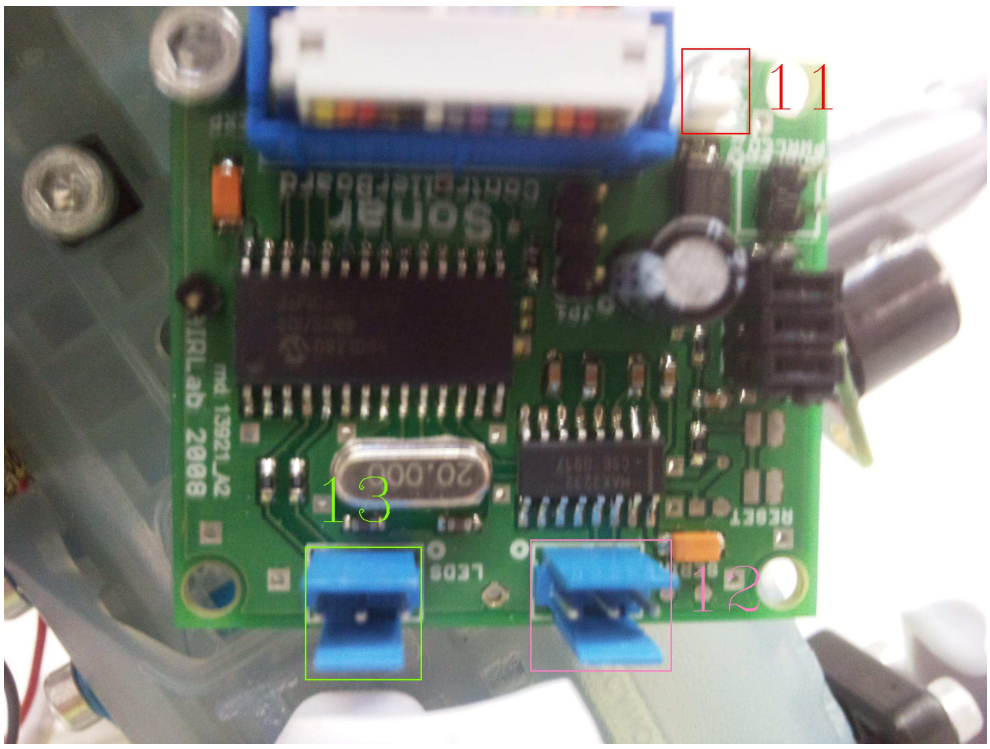

Figura B.3: Scheda sonar.

B.6 Ulteriori informazioni

Per ulteriori informazioni riguardanti il software è possibile consultare le tesi "Sviluppo di un gioco tramite l'interazione fra un robot ed il controller Wii Remote" e "RoboWII2.0: un gioco interattivo con un robot autonomo".